1.4. ВСЕПЛАТФОРМЕННАЯ РАЗРАБОТКА ИЛИ ЕСЛИ Б Я БЫЛ СУЛТАН

Недоря А.Е.., к.ф.-м.н.,

На IT-рынке идет скрытая бизнес-война между разработчиками экосистем (iOS, Android, Windows) и разработчиками приложений. Первым, для увеличения прибыли, важно ограничить (сделать уникальной) свою экосистему, вторым, для увеличения рынка и снижения расходов, делать приложения, работающие на многих (всех) платформах. Любопытно, что Google действует с двух сторон, см. Google Flutter. Нас, как людей от технологии и производства волнует не бизнес, а создание общей технологической среды, без которой затруднено (или вообще невозможно) развитие человечества. В статье, в очень краткое форме, предлагается эксперимент по переходу к изготовлению всеплатформенных программ, то есть программ, которые создаются в неком «идеально» (логическом окружении), а потом отображаются любое реальное окружение, содержащее ресурсы в достаточном качестве и количестве.

Фото самолетов: Руслан Богатырев, 2005 г., поездка с Никлаусом Виртом в музей в Монино: http://алексейнедоря.рф/?p=313

если (я султан) & первым делом(самолеты)
Эксперимент по разработке всеплатформенных программных систем.

Всеплатформенная – что это и откуда?

Развитие понятий:

- **кроссплатформенная** разработка – разрабатываем на одной платформе, переносим на другую, как правило, включает доработку исходного кода для переноса;
- **мультиплатформенная** разработка – разработка сразу идет с прицелом на несколько платформ (как правило, заранее определенных). Свежий пример: Google Flutter;
- **всеплатформенная** – следующий шаг. Разрабатываем программную систему, не привязываясь к одной или нескольким платформам, потом запускаем на любой конфигурации (совокупности платформ), в которой есть достаточно ресурсов (во всех смыслах). Обязательная составляющая часть всеплатформенности – **распределенность**.

Если мы глянем вокруг, то распределенность в наше время – это норма, а не исключение. Любое приложение на смартфоне/планшете работает с внешними сервисами. Вот только мы (по причине, полагаю, некоторой слепоты) все еще думаем о монолитных программах, работающих на одном устройстве. И при разработке рассматриваем внешнее окружение как нечто заданное, а должны рассматривать как часть разрабатываемой программной системы, как минимум, с точки зрения удовлетворения требований.

Если посмотреть с высоты птичьего полета на мой опыт разработчика/архитектора, то основное, чем я занимаюсь около 30 лет – это X-платформенность (где X = {кросс, мульт, все}). Началось с начала 1990-х, с компилирующей системы Extacy (X-to-C compilers) [Д1], далее к XDS (eXtensible Development System) [Д2, Д3, Д4], которая в своей ранней версии описана в моей кандидатской диссертации [Д5].

После этого было множество разработок и экспериментов с component-oriented programming. Я считаю компонентное, или сборочное, программирование ключевым для всеплатформенности. Это подтверждено всем моим опытом использования разных систем разработки и разработкой Вир и Вир-2 [1,2,3,4].

Так что у меня есть много, что сказать о том, как двигаться к всеплатформенности, но в этой короткой статье рассмотрим движение только с нескольких сторон, выбранных по принципу «у кого что болит».

Логическая структура и deployment

1) Разработка системы начинается с построения логической структуры, состоящей из элементов. При запуске на уровне логической структуры каждый структурный элемент работает изолированно (в песочнице/контейнере), что позволяет наблюдать/контролировать использование ресурсов каждым элементом и отслеживать взаимодействие между элементами.

2) Далее идет раскладка (deployment) логической структуры на «физическую», при этом часть элементов может оказаться на одном устройстве, в одном адресном пространстве или они могут быть соединены специализированной шиной.

Следствие: Компоненты не соединяются напрямую, а только через “разъемы”. В случае взаимодействия в рамках одного адресного пространства, обращение через разъем – это обычный вызов, в случае разных адресных пространств – RPC, возможно, с сериализацией/десериализацией, и т.д. Спецификация разъема является частью структурной компоненты, а код разъема не является частью кода компоненты. Нужный код будет подключен (выбран из набора) при раскладке на физическую структуру. Во время разработки используются разъемы, обеспечивающие максимальную наблюдаемость.

Раскладка ведется с учетом рекомендаций или ограничений. Например, если узел кластерной БД разместить на одном устройстве с обрабатывающим узлом, это может существенно повысить производительность. Какой-то вычислительный узел желательно разместить на GPU, а для другого построить специализированное устройство на ПЛИС (разработанное на Verilog’е или т.п.).

3) Независимо от раскладки в основе программной системы должны работать средства, обеспечивающие наблюдаемость, отказоустойчивость и т.п.

Логическая структура – это развитие явной схемы программы, проработанной в Вир, Вир-2 [1,5]].

Как обеспечивается работа на разных платформах.

С точки зрения Конструктора, который работает с логической структурой, все структурные элементы одинаковы. На уровне ниже они начинают разделяться на внутренние и внешние (пример внешней – БД), для которых в среде разработки (точнее, в репозитории) есть только разъемы, но не код. Внешние элементы, по сути, есть требования к физической структуре, на которую идет внедрение. Они рассматриваются как готовый узел (неизменный в рамках среды разработки). Для них не рассматривается возможность переноса на другую платформу – где стоит, там и работает.

Для внутренних элементов должна обеспечиваться возможность работы на разных платформах и в разных комбинациях – несколько компонент в одном адресном пространстве, несколько в другом адресном пространстве на том же устройстве, остальные – удаленно.

Очевидно, что есть разные способы этого достичь, в настоящее время наиболее часто используется вариант с VM. На мой взгляд, это не наш способ. Дополнительная прослойка (VM) вносит дополнительные сложности. Она была бы необходима, если бы мы не могли обойтись без чего-то вроде JVM, но мы уже давно можем. См, например, [6] про VM и историю развития компиляторов.

Если убрать лишние костыли и работать с нативным кодом, то я вижу это так:

- В репозитории хранятся элементы в виде промежуточного кода (например, LLVM IR). По запросу к репозиторию выдается бинарный код для конкретной платформы в нужном формате.
- Полагаю, что в боевом варианте должен быть кэш бинарного кода, чтобы не оптимизировать/генерировать часто используемый код. В экспериментальной версии можно обойтись без кэша.
- Кроме элементов (компонент) в репозитории должен быть код разъемов и **код переходников** (для переходников хранится готовый код для каждой платформы).

Переходники – это очень важная часть эксперимента, но так как я уже, в общем, писал о них см. [1], не буду повторяться.

Замечание: LLVM IR – не самое удобное представление, так как оно сильно завязано на специфику платформы (TargetMachine).

В рамках экспериментальной Вир-2 я использую LLVM, в дальнейшем надо будет подумать о переходе на другое внутреннее представление.

Языки программирования

Question: Do we "really" need more programming languages?
Alan Kay: We could use a few "good ones" (meaning ones that really are about the realities, needs and scales of the 21st century).
Q: Could you list top 3 good ones as per your opinion?

> **Alan Kay**: I meant, we could "use three good ones", not that I knew of three ...
>
> Hacker News, June 20, 2016

Все же, какие языки программирования использовать для всеплатформенного программирования?

Я думал об ответе на этот вопрос, см. например [2,3,4], но пока не нашел работающего ответа, который должен включать: языки + компиляторы + toolchain + интеграция с другими языками/библиотеками. Ответ на вопрос об используемых языках программирования должен быть одной из целей эксперимента.

Оптимизация

Программная система, собранная в логическую структуру, почти всегда будет не оптимальна по производительности/использованию ресурсов. И это понятно, так как она должна быть удобна, в первую очередь, для проектирования, прототипирования, внесения изменений, тестирования и отладки верхнего уровня.

А дальше нужны оптимизации, и они могут отличаться от привычных. Например:

- оптимизация через агрегацию – несколько компонент “объявляются” компонентой более высокого уровня. Код такой компоненты генерится целиком, соответственно, при этом работают компиляторные оптимизации.
- в каких-то случаях может использоваться противоположное действие, назовем его “сегментация”. Компонента разделяется на несколько. Например, выделяется подкомпонента и она горизонтально масштабируется.
- оптимизация раскладки – требование или рекомендация “близкого” расположения нескольких компонент. “Близко” может иметь разный смысл: на одном устройстве, в локальной сети, и т.п.
- оптимизация через конкретизацию. Например, известно, что компонента всегда работает на устройствах одной платформы, тогда можно заменить часть более универсального кода на менее универсальный.

И так далее.

Замечу, что оптимизации не должны приводит к разрушению логической структуры.

Что еще надо прорабатывать?

Перечислю то, что на поверхности.

Компоненты:

- язык описания бинарного интерфейса
- иммутабельность интерфейса (copy on write для компоненты при изменении интерфейса)
- версионность
- наличие нескольких реализаций (optimized for speed/memory/flexibility)

UI:

- выделение UI компонент
- возможность задания нескольких UI. Например, GUI + голос + запахи (вроде бы шутка)
- Адаптивные GUI

Управление ресурсами, в первую очередь, ОЗУ:

- component-based, thread-based memory management
- взаимодействие компонент с разным подходом к освобождению ресурсов (GC, non-GC)

Мета-средства:

- динамическая типизация и динамическое знакомство. Оптимизация динамики в статику.

Часть из этих тем затронута мной в статьях блога, см. алексейнедоря.рф

Как видно, объем работы такой, что одного султана недостаточно, нужно 3, а еще лучше 33 султана.

Вместо послесловия

> ….in programming there is a wide-spread 1st order theory that one shouldn't build one's own tools, languages, and especially operating systems. This is true—an incredible amount of time and energy has gone down these ratholes. On the 2nd hand, if you can build your own tools, languages and operating systems, then you ***absolutely should*** because the leverage that can be obtained (and often the time not wasted in trying to fix other people's not quite right tools) can be incredible.
>
> Alan Kay “The Power of Context”, 2004

Ссылки на статьи автора

[1] “Технология разработки мультиплатформенных программ на основе явных схем программ”, http://digital-economy.ru/stati/tekhnologiya-razrabotki-multiplatformennykh-programm-na-osnove-yavnykh-skhem-programm

[2] “Триада языков программирования”, http://xn--80aicaaxfgwmwf3q.xn--p1ai/?p=298

[3] “Компонентный ассемблер для цифрового пространства. Часть 1”, http://digital-economy.ru/stati/komponentnyj-assembler-dlya-tsifrovogo-prostranstva
[4] “Компонентный ассемблер для цифрового пространства. Часть 2. Дух языка”, http://digital-economy.ru/stati/komponentnyj-assembler-chast-2-dukh-yazyka
[5] “Вир. Явная схема программы”, http://xn--80aicaaxfgwmwf3q.xn--p1ai/?p=269
[6] “Забытое 40 лет назад новое и как оно может изменить нашу жизнь”, http://xn--80aicaaxfgwmwf3q.xn--p1ai/?p=255

Дополнительные материалы

[Д1] “Extacy - система разработки ПО на языках Модула-2 и Оберон-2”, http://www.kronos.ru/literature/extacy
[Д2] GitHub: “XDS Modula-2/Oberon-2 Development System for Windows”, https://github.com/excelsior-oss/xds
[Д3] Е. Налимов, А. Недоря, “Перенацеливаемый оптимизирующий Модула-2/Оберон-2 компилятор”, “Технология программирования, 1995, http://oberon2005.oberoncore.ru/paper/obe_ned.pdf
[Д4] V.Mikheev, “Design of Multilangual Retargetable Compilers: Experience of XDS Framework Evolution”, 1999, http://oberon2005.oberoncore.ru/paper/vm1999.pdf
[Д5] А. Недоря, “Расширяемая переносимая система программирования, основанная на биязыковом подходе”, http://www.kronos.ru/literature/nedorya/abstrac

References in Cyrillics

[1] “Tekhnologiya razrabotki mul'tiplatformennyh programm na osnove yavnyh skhem programm”, http://digital-economy.ru/stati/tekhnologiya-razrabotki-multiplatformennykh-programm-na-osnove-yavnykh-skhem-programm
[2] “Triada yazykov programmirovaniya”, http://xn--80aicaaxfgwmwf3q.xn--p1ai/?p=298
[3] “Komponentnyj assembler dlya cifrovogo prostranstva. Chast' 1”, http://digital-economy.ru/stati/komponentnyj-assembler-dlya-tsifrovogo-prostranstva
[4] “Komponentnyj assembler dlya cifrovogo prostranstva. Chast' 2. Duh yazyka”, http://digital-economy.ru/stati/komponentnyj-assembler-chast-2-dukh-yazyka
[5] “Vir. Yavnaya skhema programmy”, http://xn--80aicaaxfgwmwf3q.xn--p1ai/?p=269
[6] “Zabytoe 40 let nazad novoe i kak ono mozhet izmenit' nashu zhizn'”, http://xn--80aicaaxfgwmwf3q.xn--p1ai/?p=255
[D3] E. Nalimov, A. Nedorya, “Perenacelivaemyj optimiziruyushchij Modula-2/Oberon-2 kompilya-tor”, “Tekhnologiya programmirovaniya, 1995, http://oberon2005.oberoncore.ru/paper/obe_ned.pdf
[D5] A. Nedorya, “Rasshiryaemaya perenosimaya sistema programmirovaniya, osnovannaya na biyazykovom podhode”, http://www.kronos.ru/literature/nedorya/abstrac

Алексей Евгеньевич Недоря (a.nedoria@astra-vir.ru)

Ключевые слова: технология разработки программ; кроссплатформенное программирование, мультиплатформенное программирование; языки программирования.

Alex Nedorya, All-platform programs development or if I was a billionaire

Keywords: software development technology, crossplatform programming, multiplatform programing, distributed programs, programming languages.

Abstract. In the IT market there is a hidden business war between ecosystem manufacturers (iOS, Android, Windows) and application developers. For first, to increase profits, it is important to bound (make unique) their ecosystems, for second, to increase the market and to reduce costs, it is important to make applications running on many (all) platforms. Curiously, Google acts on both sides (see Google Flutter). We, as technology and production people, are not concerned about business, but about the creation of a common technological environment, without which the development of humankind is difficult (or impossible). The article, in a very brief form, proposes an experiment on the transition to the production of all-platform programs, that is, programs that are created in a certain "ideal" (logical environment), and then are deployed to any real environment that contains resources in sufficient quality and quantity.

DOI: 10.34706/DE-2019-02-04